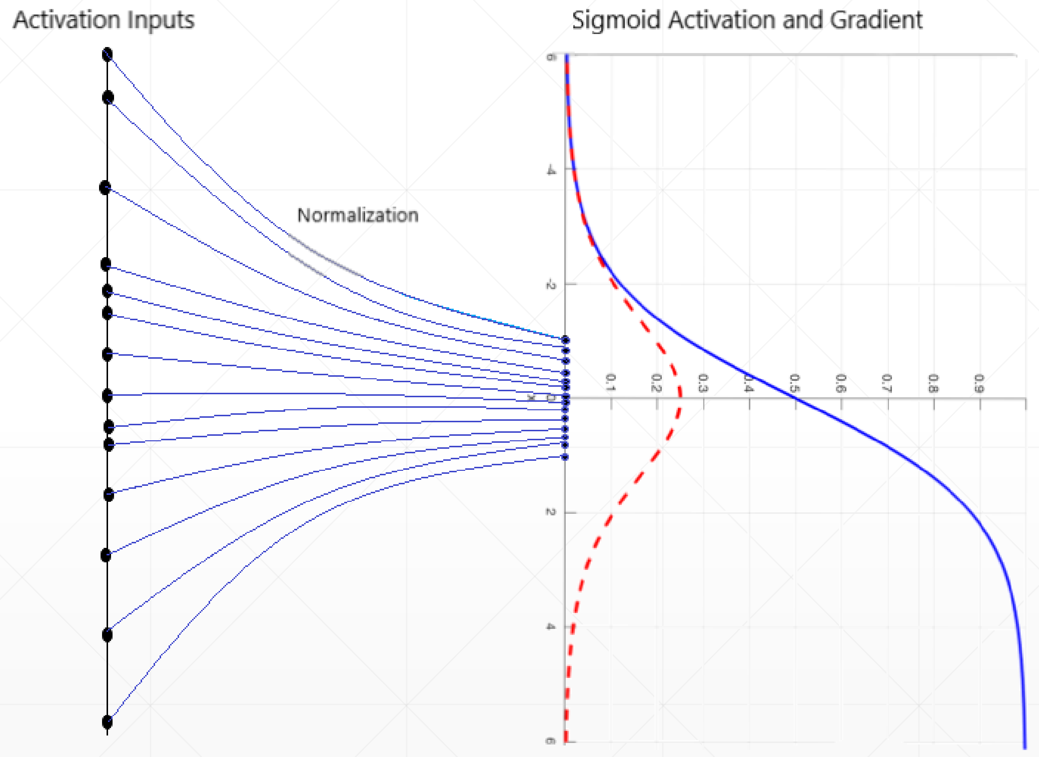
PyTorch

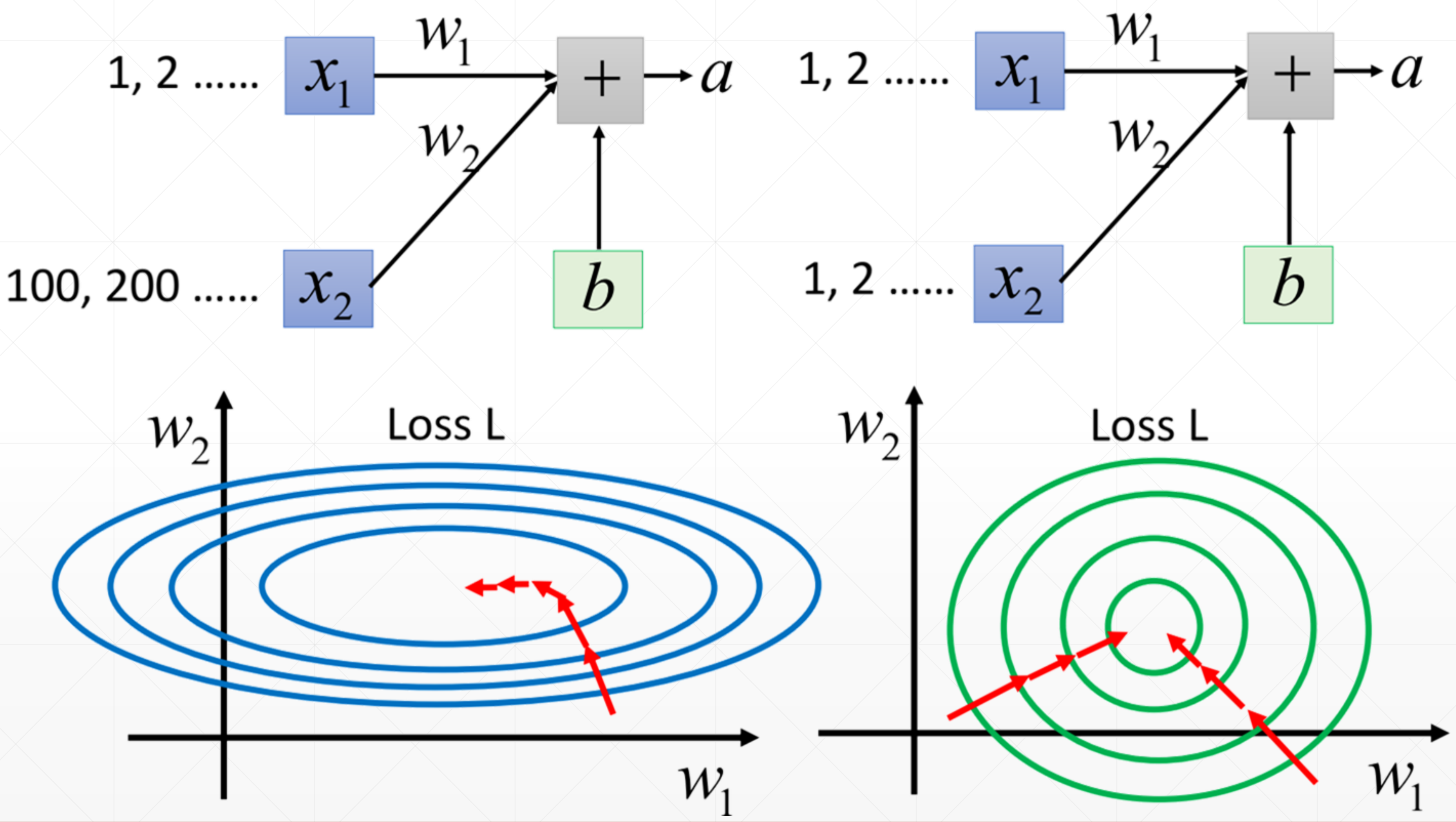
Batch Norm

主讲人：龙良曲

Intuitive explanation

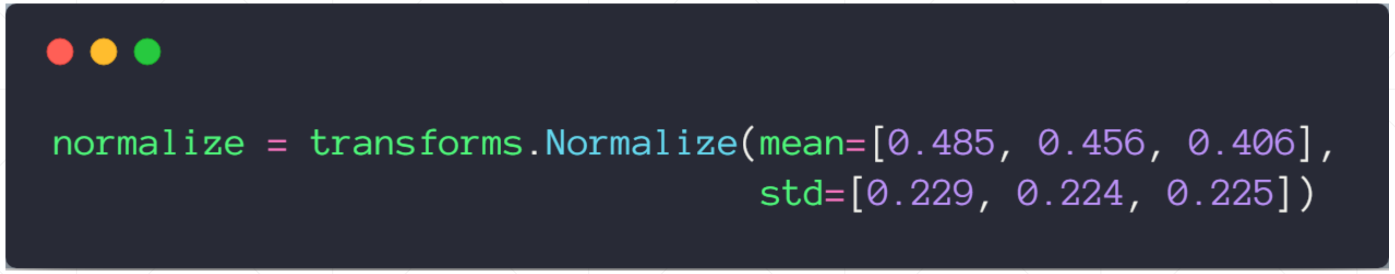


Intuitive explanation



Feature scaling

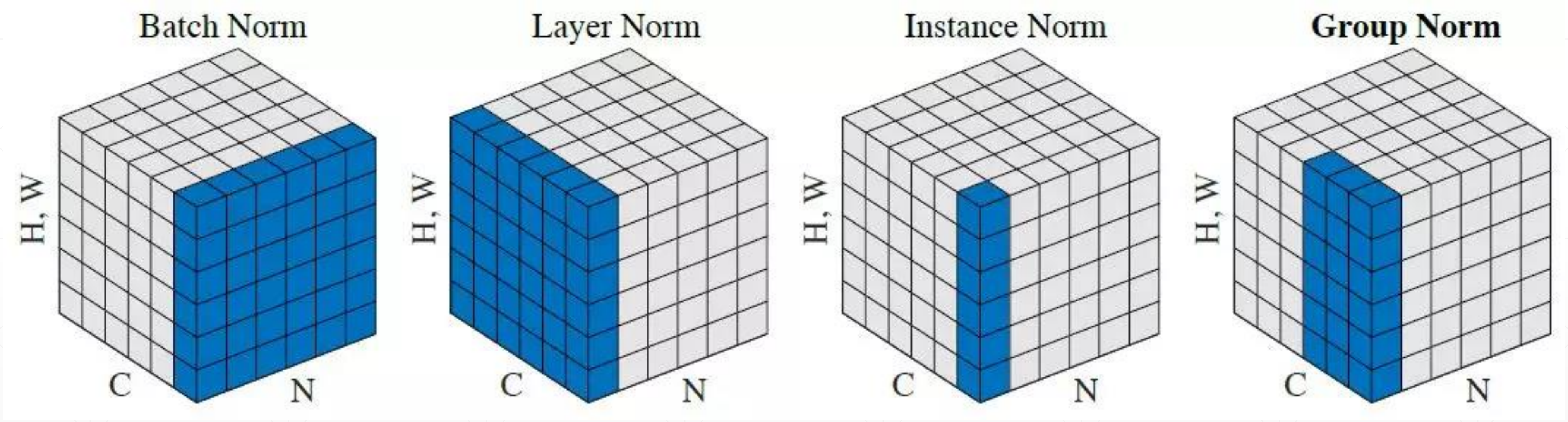
- Image Normalization



```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

- Batch Normalization

Batch Norm



C

N

N,C

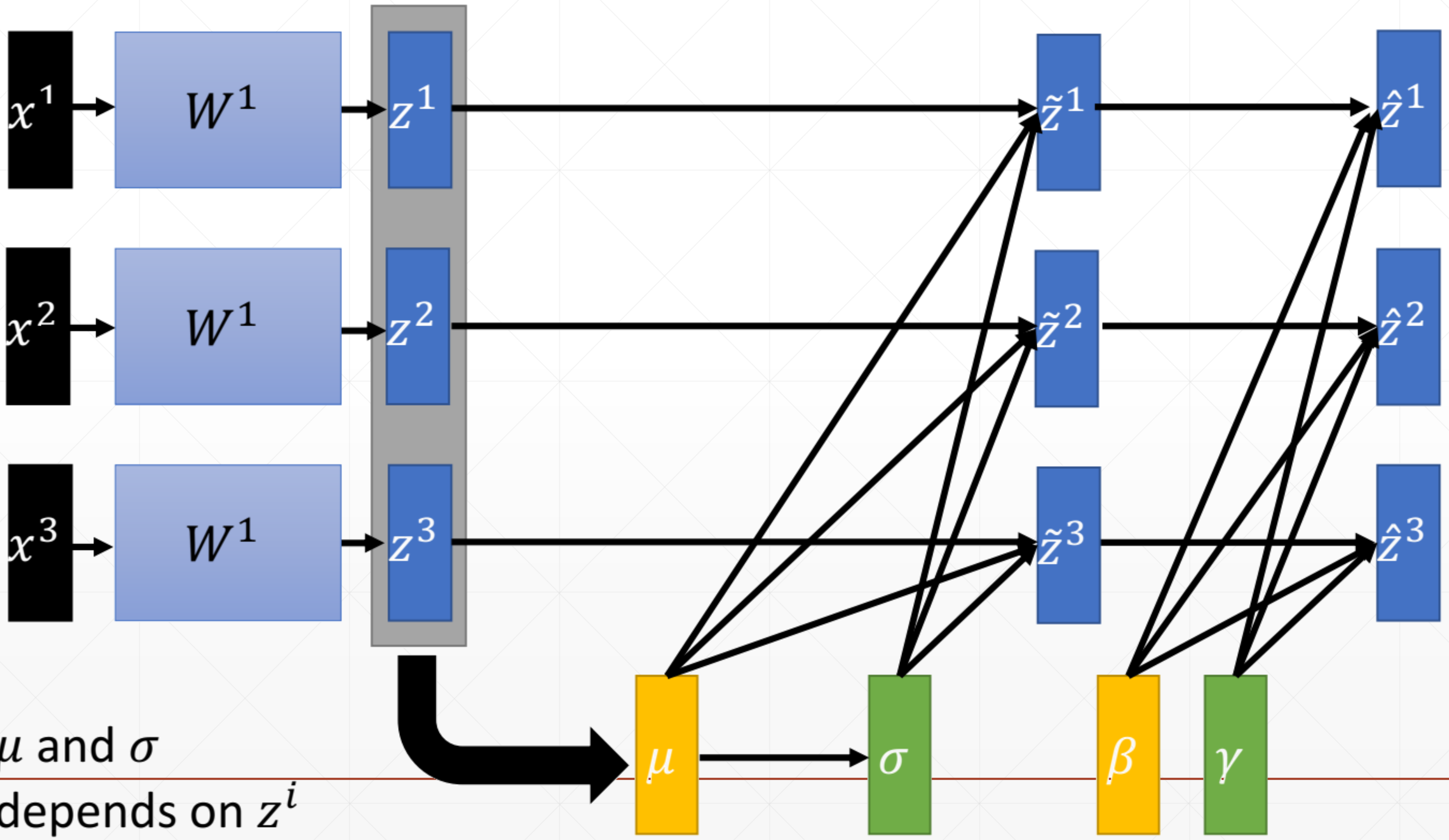
N,xC

https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

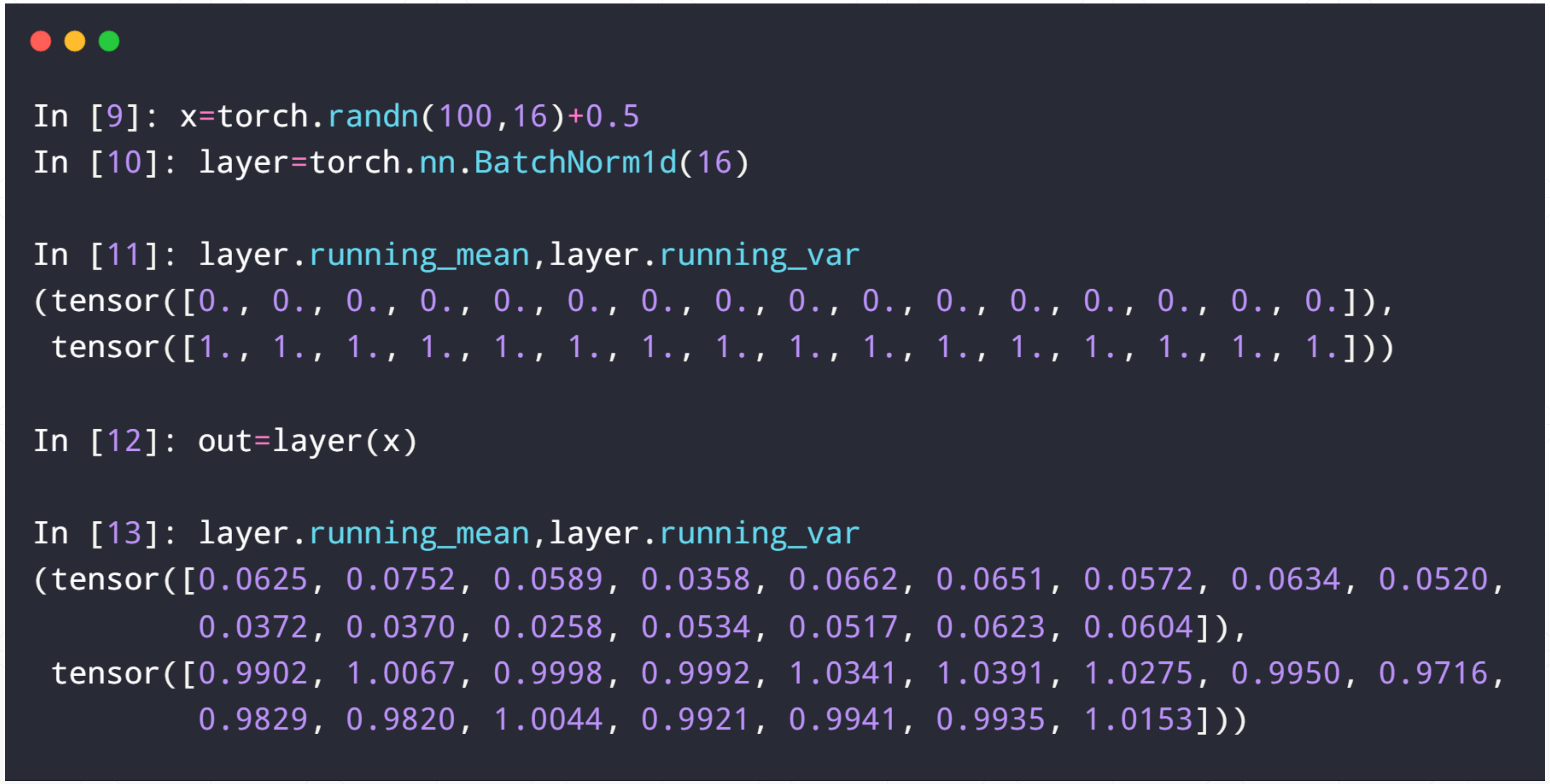
Batch normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

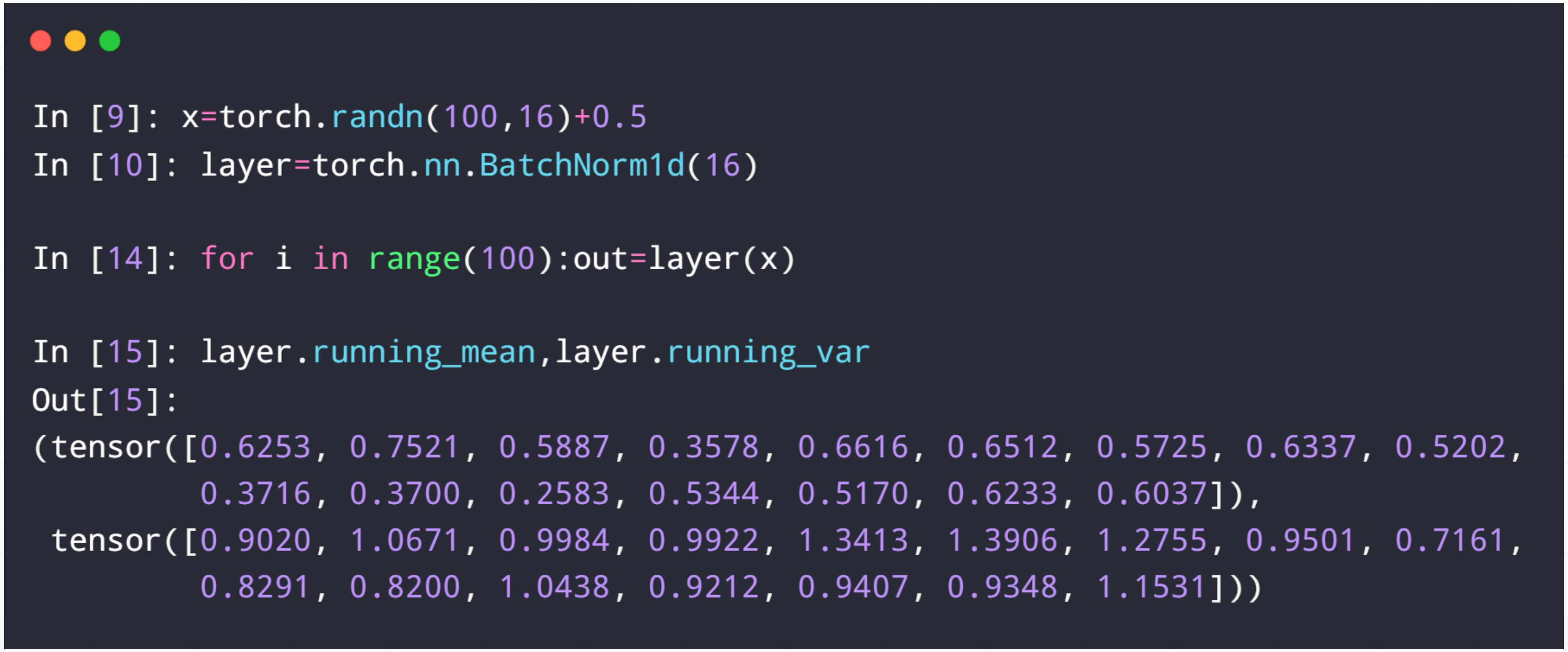
$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$



μ and σ depends on z^i

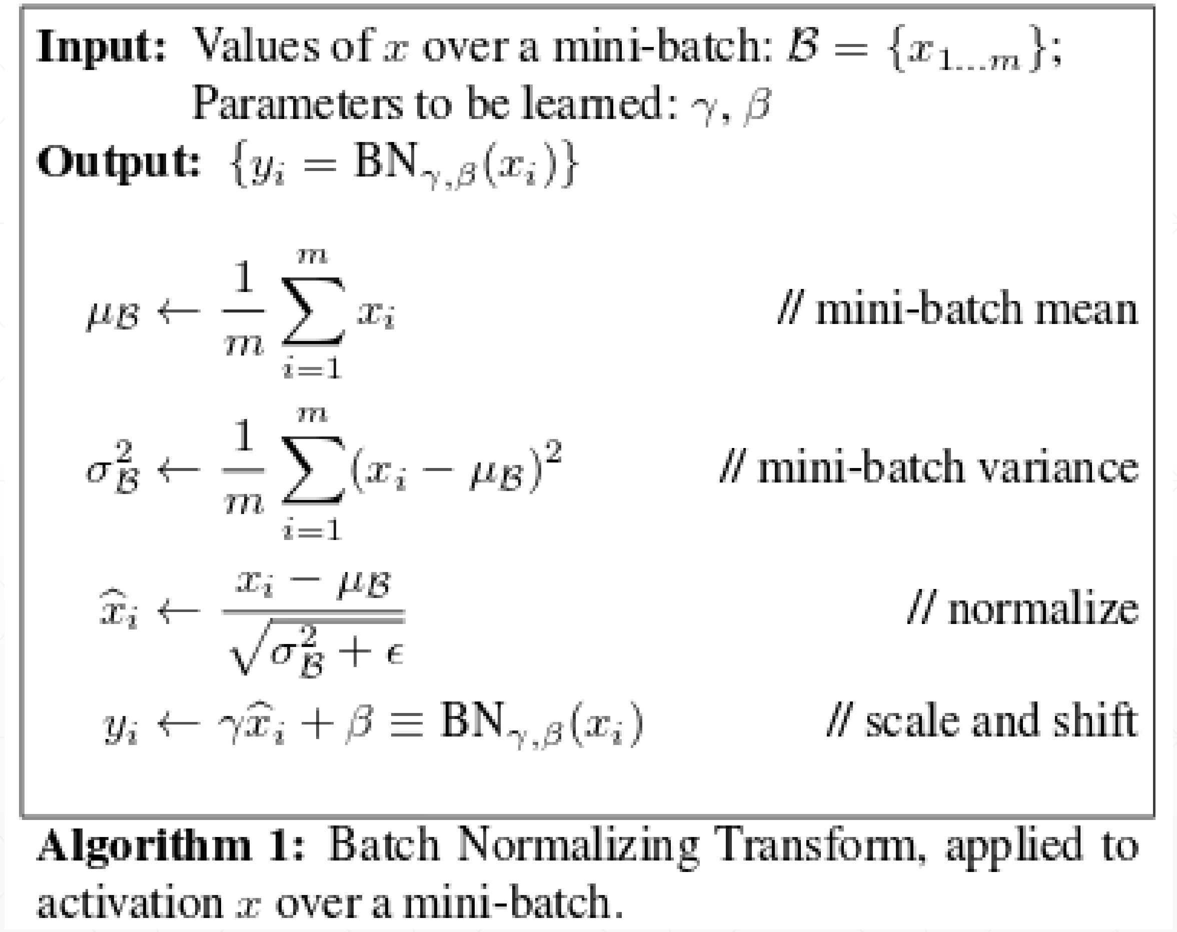


```
In [9]: x=torch.randn(100,16)+0.5
In [10]: layer=torch.nn.BatchNorm1d(16)

In [11]: layer.running_mean,layer.running_var
(tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]))

In [12]: out=layer(x)

In [13]: layer.running_mean,layer.running_var
(tensor([0.0625, 0.0752, 0.0589, 0.0358, 0.0662, 0.0651, 0.0572, 0.0634, 0.0520,
         0.0372, 0.0370, 0.0258, 0.0534, 0.0517, 0.0623, 0.0604]),
 tensor([0.9902, 1.0067, 0.9998, 0.9992, 1.0341, 1.0391, 1.0275, 0.9950, 0.9716,
         0.9829, 0.9820, 1.0044, 0.9921, 0.9941, 0.9935, 1.0153]))
```



```
In [9]: x=torch.randn(100,16)+0.5
In [10]: layer=torch.nn.BatchNorm1d(16)

In [14]: for i in range(100):out=layer(x)

In [15]: layer.running_mean,layer.running_var
Out[15]:
(tensor([0.6253, 0.7521, 0.5887, 0.3578, 0.6616, 0.6512, 0.5725, 0.6337, 0.5202,
         0.3716, 0.3700, 0.2583, 0.5344, 0.5170, 0.6233, 0.6037]),
 tensor([0.9020, 1.0671, 0.9984, 0.9922, 1.3413, 1.3906, 1.2755, 0.9501, 0.7161,
         0.8291, 0.8200, 1.0438, 0.9212, 0.9407, 0.9348, 1.1531]))
```

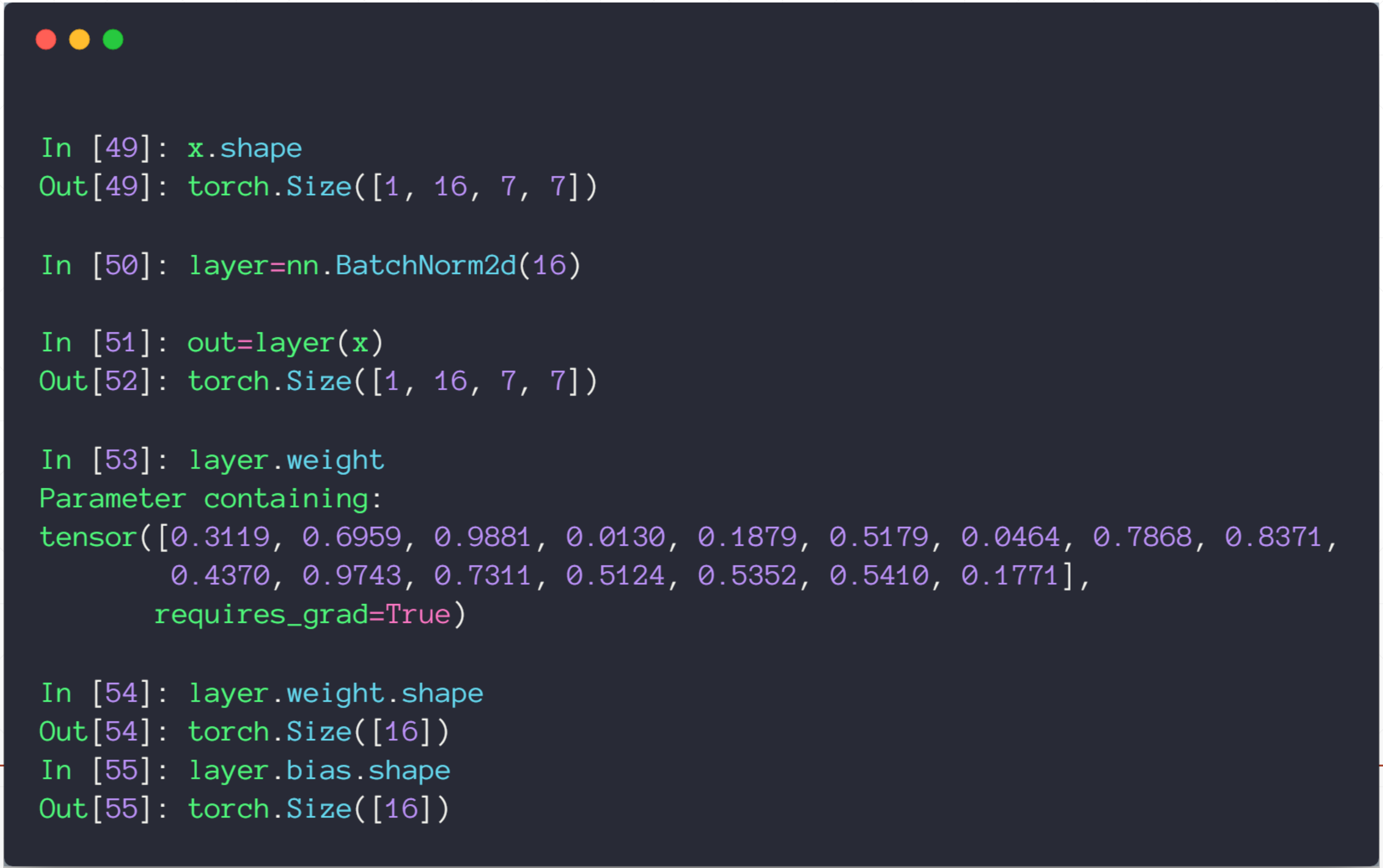
Pipeline



```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                        // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$      // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                  // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$        // scale and shift
```

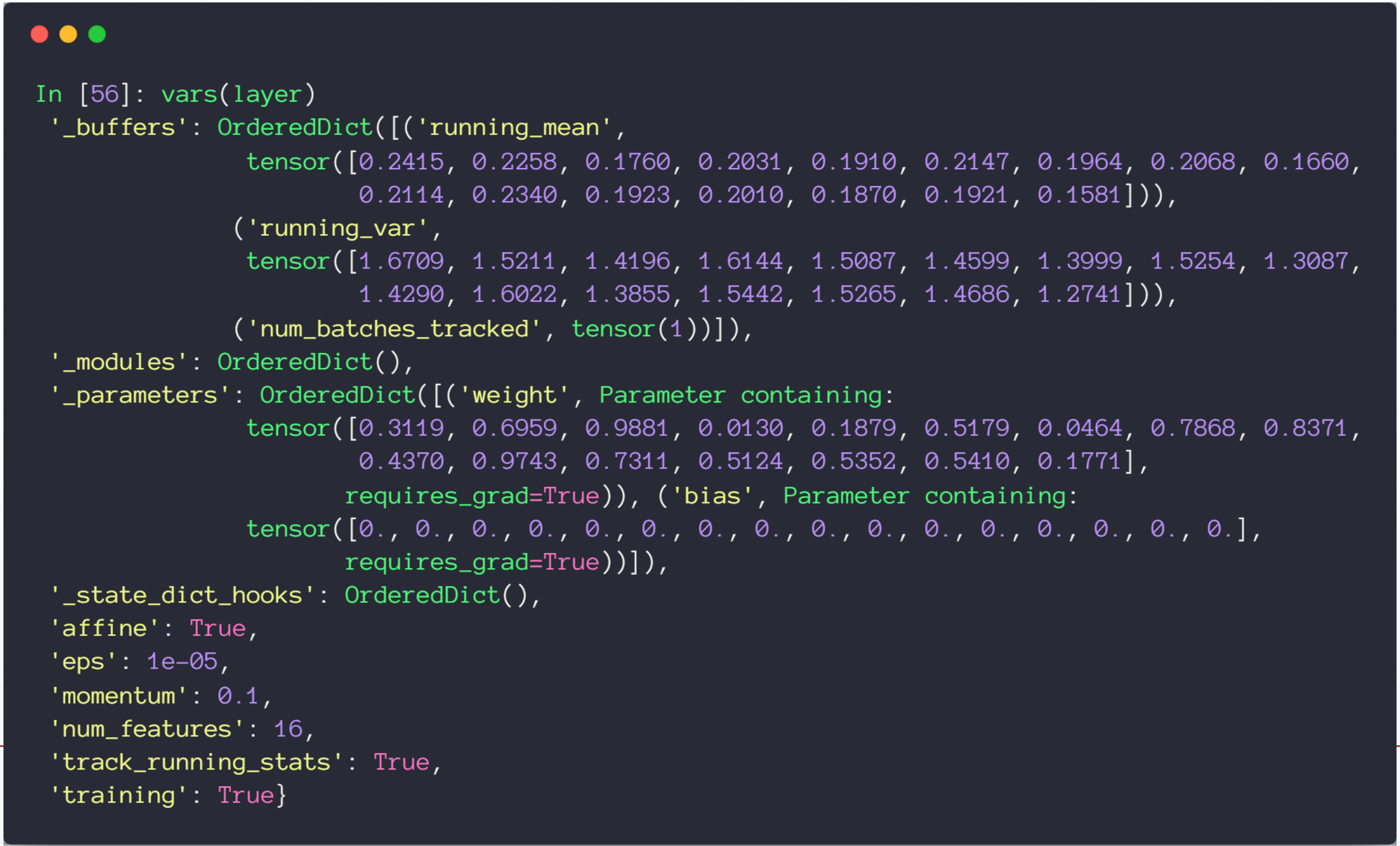
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

nn.BatchNorm2d



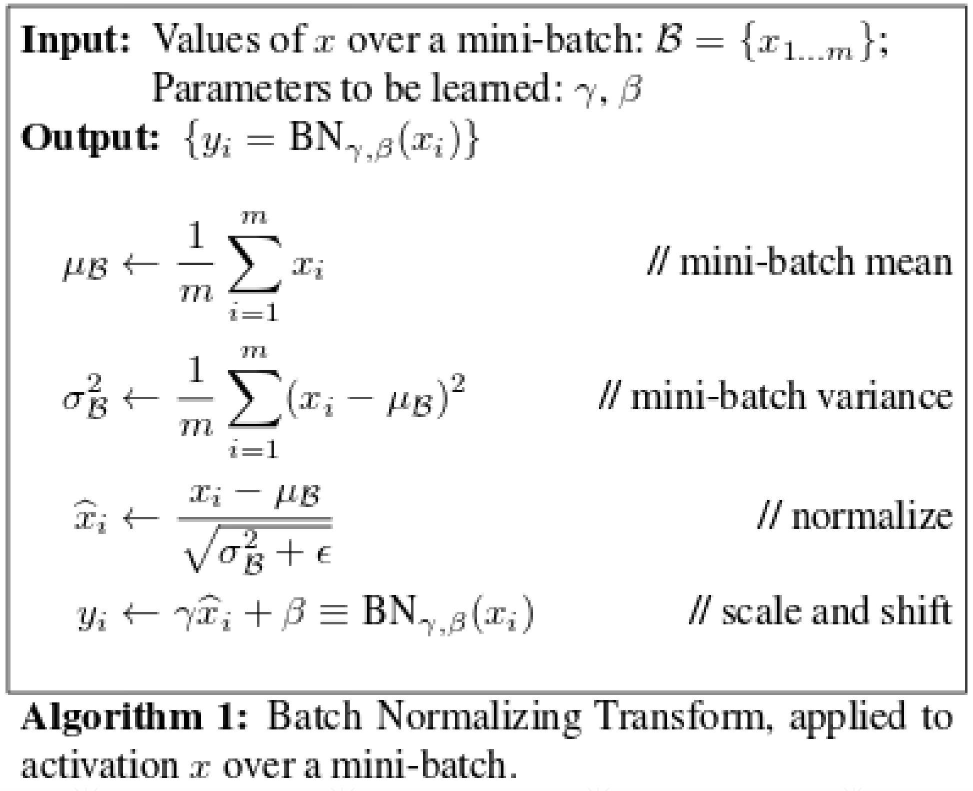
```
In [49]: x.shape
Out[49]: torch.Size([1, 16, 7, 7])

In [50]: layer=nn.BatchNorm2d(16)

In [51]: out=layer(x)
Out[52]: torch.Size([1, 16, 7, 7])

In [53]: layer.weight
Parameter containing:
tensor([0.3119, 0.6959, 0.9881, 0.0130, 0.1879, 0.5179, 0.0464, 0.7868, 0.8371,
        0.4370, 0.9743, 0.7311, 0.5124, 0.5352, 0.5410, 0.1771],
       requires_grad=True)

In [54]: layer.weight.shape
Out[54]: torch.Size([16])
In [55]: layer.bias.shape
Out[55]: torch.Size([16])
```

Class variables



```
In [56]: vars(layer)
 '_buffers': OrderedDict([('running_mean',
               tensor([0.2415, 0.2258, 0.1760, 0.2031, 0.1910, 0.2147, 0.1964, 0.2068, 0.1660,
                       0.2114, 0.2340, 0.1923, 0.2010, 0.1870, 0.1921, 0.1581])),
              ('running_var',
               tensor([1.6709, 1.5211, 1.4196, 1.6144, 1.5087, 1.4599, 1.3999, 1.5254, 1.3087,
                       1.4290, 1.6022, 1.3855, 1.5442, 1.5265, 1.4686, 1.2741])),
              ('num_batches_tracked', tensor(1))]),
 '_modules': OrderedDict(),
 '_parameters': OrderedDict([('weight', Parameter containing:
               tensor([0.3119, 0.6959, 0.9881, 0.0130, 0.1879, 0.5179, 0.0464, 0.7868, 0.8371,
                       0.4370, 0.9743, 0.7311, 0.5124, 0.5352, 0.5410, 0.1771],
                      requires_grad=True)), ('bias', Parameter containing:
               tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
                      requires_grad=True))]),
 '_state_dict_hooks': OrderedDict(),
 'affine': True,
 'eps': 1e-05,
 'momentum': 0.1,
 'num_features': 16,
 'track_running_stats': True,
 'training': True}
```

Test



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

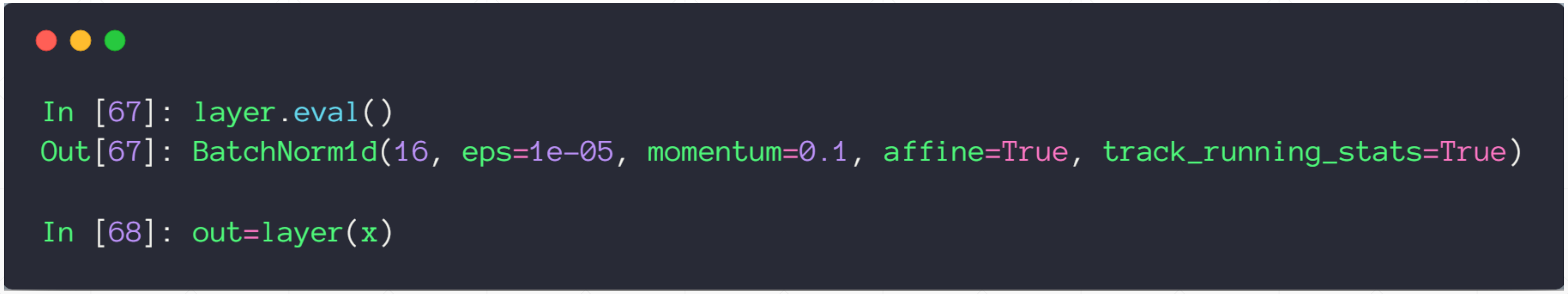
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

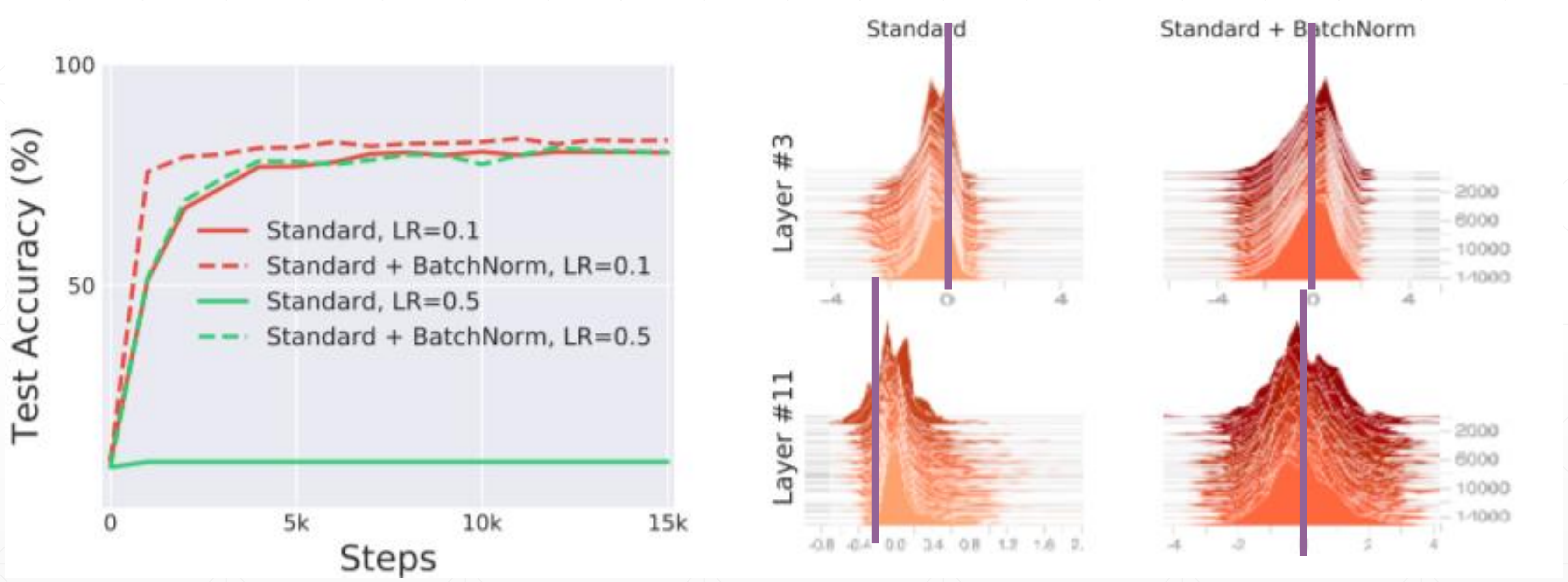
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.



```
In [67]: layer.eval()
Out[67]: BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

In [68]: out=layer(x)
```

Visualization



Advantages

- Converge faster
- Better performance
- Robust
  - stable
  - larger learning rate

下一课时

经典CNN

Thank You.